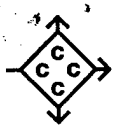

Computer Command and Control Company

Software Tools for Formal Specification and Verification of Distributed Real-Time Systems

Final Report

Submitted to:
Office of Naval Research
Ballston Tower One
800 N. Quincy Street
Arlington, VA 22217-5660

Prepared Under SBIR Phase II
Contract No. N00014-95-C-0131
Data Item No. A002

September 30, 1997

Computer Command and Control Company
2300 Chestnut Street, Ste. 230
Philadelphia, PA 19103

Contents

Chapter I

Executive Summary

I-1 Objectives

This is the final report prepared under SBIR Phase II Contract N00014-95-C-0131 from the Office of Naval Research. It reports on the development of a set of software tools for specification and verification of distributed real time systems using formal methods. The task of this SBIR Phase II effort was to create a commercial-strength CASE toolset suitable for handling real-life verification problems. A preceding Phase I contract was concerned with development of the approach to the problem and design of a prototype environment [14]. Forthcoming Phase III work will demonstrate the utility of the toolset to potential customers and establish it as a commercial off-the-shelf (COTS) specification and verification product. It is important to note that commercialization work, which is the task of Phase III efforts, has already begun during the current Phase II period (see Section IV).

The toolset has been given the name PARAGON, which stands for "**P**rocess-**A**lgebraic **R**eal-time **A**nalysis with **G**raphics-**O**riented **N**otation". The toolset is indented to be used by designers of real-time systems for early detection of errors. The mathematical complexity of formal specification and verification has been hidden from the end users as much as possible. To achieve this, the specification language uses notions used by designers in their work as primitives. This provides for concise specifications, readable even by a non-specialist.

I-2 The Problem and Approach

The use of computers in mission-critical applications is now firmly established. Moreover, important control tasks such as commonly found in Navy systems are performed by networks of computers, often spatially separated by considerable distances. In addition, such tasks have very stringent timing requirements. Therefore, many mission-critical applications inherently belong to the class of distributed real-time systems. Designing correct distributed real-time programs proved to be very difficult in practice. Two aspects of such systems contribute to its complexity:

1. A distributed system has several components running asynchronously in parallel. Events in different components can be arbitrarily interleaved, which may lead to subtle communication problems between components.

2. On top of that, correctness of a real-time system's behavior depends on delays between events in components. Difference in relative speeds of components can lead to timing errors that are difficult to reproduce and find through traditional validation methods like simulation.

In the future, the size of mission-critical systems will inevitably increase, thus making the task of establishing correct behavior by traditional methods ever more difficult.

Verification approaches based on formal methods provide a mathematically sound way to analyze the system design for undesirable behaviors. Several specification and verification toolsets for analysis of distributed and real-time systems based on a wide variety of formal methods has been put forward in recent years. All of these tools suffer from some or all of the following problems:

- specifications are hard to understand for engineers and require a formal methods expert to construct them;

- they are not scalable enough to tackle real-life problems;

We made special efforts to address these problems in our design, as described below.

The use of intuitive constructs. We make specifications more understandable to non-specialists by introducing special constructs that capture notions commonly used in real-time systems design. The notion of a *resource* is central in many real-time applications, where processes compete for access to a set of shared resources.

Resource contention is normally resolved through the use of priorities. Resources, to the best of our knowledge, have never been considered as first-class notions in specifications. The treatment of priority is also absent from most commonly usd formal methods. In some cases, the use of resources and priorities can be simulated by other means of a formal language. However, this leads to counter-intuitive and less concise specifications.

Graphical and textual interfaces. It is known that high-level descriptions of a system can easily be represented graphically, giving the user a clear understanding of interconnections between components in a single glance. On the other hand, lower level specifications may be very cumbersome in a graphical form. In that case, a textual specification may be preferrable, both in terms of readability and the time it takes to construct the specification. Finally, different users may have different personal preferences.

PARAGON supports specifications in both graphical and textual formats. Both specification languages have formal semantics. Moreover, semantics for the two languages are compatible, which allows one to mix the two languages in a large specification as appropriate.

Design for scalability. Our specification language was developed with scalability in mind. They offer constructs for hierarchical structuring of specifications, and offer facilities for top-down design through refinement.

The use of notions specific to the problem domain, especially the notion of a resource, contributes to scalability of specifications. The ability to refine resources into process specifications allows the user to perform verification on an abstract level, thus working with specifications of much smaller sizes.

Design for extension. PARAGON is designed as an open system which can be easily modified to include the latest developments in specification and verification technology. The tools are designed using the object-oriented framework in a modular fashion that facilitates enhancements to the code. A guide for future developers [12] is made available, describing the internal structure of the toolset in detail. This reduces maintenace costs for PARAGON, and ensures that the toolset remains up-to-date for a long time.

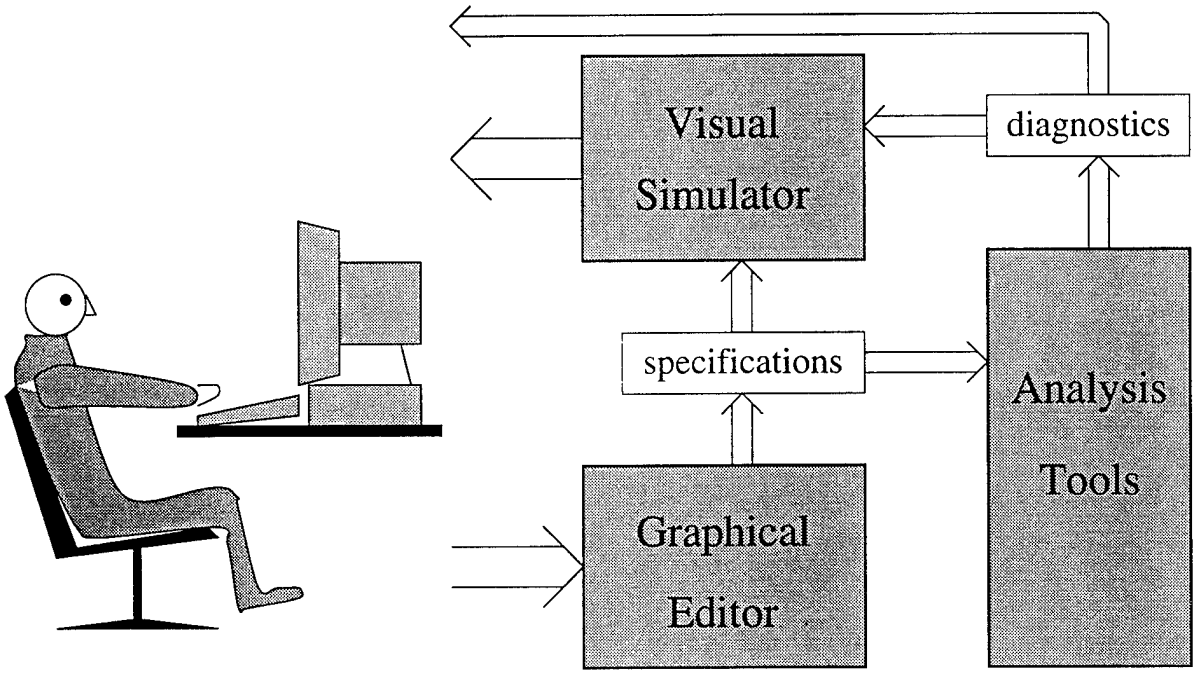

Figure I-1: Structure of PARAGON toolset.

I-3 Structure of PARAGON toolset

PARAGON currently contains three main tools:

- The graphical editor for visual specifications;
- The visual simulator;
- The back-end verification tool VERSA.

The structure of the toolset is represented in Figure I-1. Specifications constructed via the graphical editor may be either verified using VERSA, or simulated using the simulator tool. The back-end verification tool performs analysis and presents the user with the outcome. If verification fails, the verification tool produces diagnostic information that is used to find the problem. The diagnostic information takes the form of a trace that can be used to drive the simulator, thus helping to locate the the problem.

The tools are described in more detail in Chapter II.

I-4 Verification Methodology of PARAGON

A successful CASE tool must be based on a specific methodology, compatible with existing software engineering practices. Designing PARAGON, we paid careful attention to development of a verification methodology that would make PARAGON practically usable.

In this methodology, we start by constructing a specification of the system on the desired level. Depending on what is provided by the user, the specification can be created from an informal description of the system, pseudo-code, or the actual code of the system. In addition, we construct a separate specification for each requirement for the system. Requirements are usually represented informally, using English language. We have found that this is the most time-consuming part of the specification process, which involves extensive communication with the user until requirements are precise enough to be transformed into a formal representation.

Most requirements are represented as tester processes that are composed in parallel with the system specification. A tester for a requirement watches for system behaviors that violate the requirement, and signals a failure if it detects one.

The composite specification is given as input to VERSA, which looks for reachability of the failure event (alternatively, one can look for deadlocks introduced by the tester after the failure). If there are reachable failures, VERSA produces an execution trace leading to the undesirable event. This trace is fed into the simulator, which visualizes the execution for the user, thus helping the user to discover the source of the problem.

This technology has proved successful in benchmark applications of PARAGON.

Chapter II

Overview of Developed Products

PARAGON toolset consists of three main parts: a graphical editor, a visual simulator and a verification back-end tool. The first two tools work with specifications expressed in visual specification language GCSR. Verification engine VERSA is developed mostly by Real-Time Systems Group at University of Pennsylvania headed by Professor Insup Lee. VERSA supports specifications expressed in ACSR, a process-algebraic formalism for specification of real-time distributed systems [18]. There are translations between ACSR and GCSR, which allow these tools to be connected and used together in a toolset. Visual primitives of GCSR provide for readability and maintainability of specifications, while ACSR, with its simple textual format, facilitates design of verification algorithms.

II-1 The Graphical Editor

The graphical editor for the PARAGON toolset allows users to design specifications in GCSR language. The language has process-algebraic semantics that allows formal verification to be performed on GCSR specifications. The editor contains a drawing area for displaying specifications, a toolbar featuring a set of drawing and editing modes. The user interface of the GCSR editor is shown in figure II-1. This section gives a brief outline of the editor's functionality. Detailed description can be found in the User's Manual for PARAGON toolset. [13].

In addition to drawing and visual editing, the GCSR editor allows the user to check resulting specifications for consistency, save them in a file, and also establishes connection with the verification tool. The editor supports top-down hierarchical de-

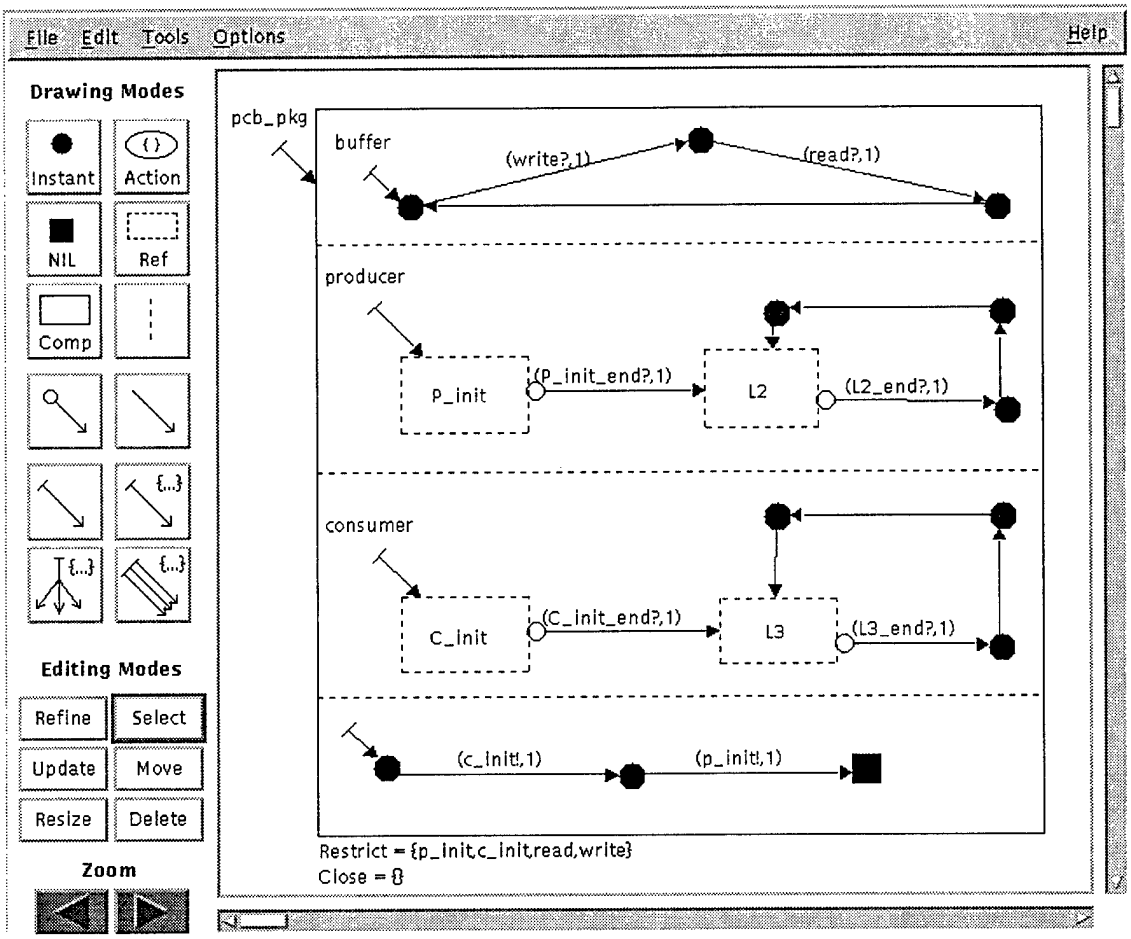

Figure II-1: User Interface of the Editor

velopment of complex specifications via property-preserving refinement. Modularity of specifications is enabled in GCSR language through process references. The editor provides further support for modularity through secondary editing windows so that designers may view multiple subspecifications together. The tool offers help in locating the definition for a process reference, bringing it up as a secondary window.

II-2 The Visual Simulator

The simulator for GCSR specifications is a separate tool in the PARAGON toolset. It offers automatic or user-guided execution of GCSR specifications. The display facility of the simulator is the same as that in the editor tool, ensuring the same look-and-feel for all GCSR-based tools. Simulation proceeds directly on the specification, and all simulation events are highlighted on the screen. At every step, the simulator offers the updated set of enabled transitions in a separate window. Another window displays the accumulated history prior simulation events (an execution trace). By selecting an element in the trace, the user can return to an arbitrary point in the simulation history. The user interface of the simulator is shown in Figure II-2. Detailed description of the simulator functionality can be found in the User's Manual for PARAGON toolset [13].

The simulator offers two simulation modes: *step* mode and *automatic* mode. In the step mode, the user can select the next transition to be executed from the list of currently enabled transitions, constructed and updated by the simulator after each step. In the automatic mode, the next transition for execution is chosen at random. The automatic mode allows the user to set breakpoints. A breakpoint may be a *partial* state, i.e. the state of only a subset of processes in the specification. When a breakpoint is reached, the simulation is interrupted and the user can inspect the current state and the execution trace leading to that state.

Since a specification can be structured over several files, secondary simulator windows are opened automatically as needed. All secondary simulation windows are parts of the same simulation process and share the simulation state. A breakpoint, for example, can contain nodes from several different windows.

The user can start simulation from an arbitrary state by entering the desired state manually. The objects comprising the new state are selected on the screen with mouse clicks.

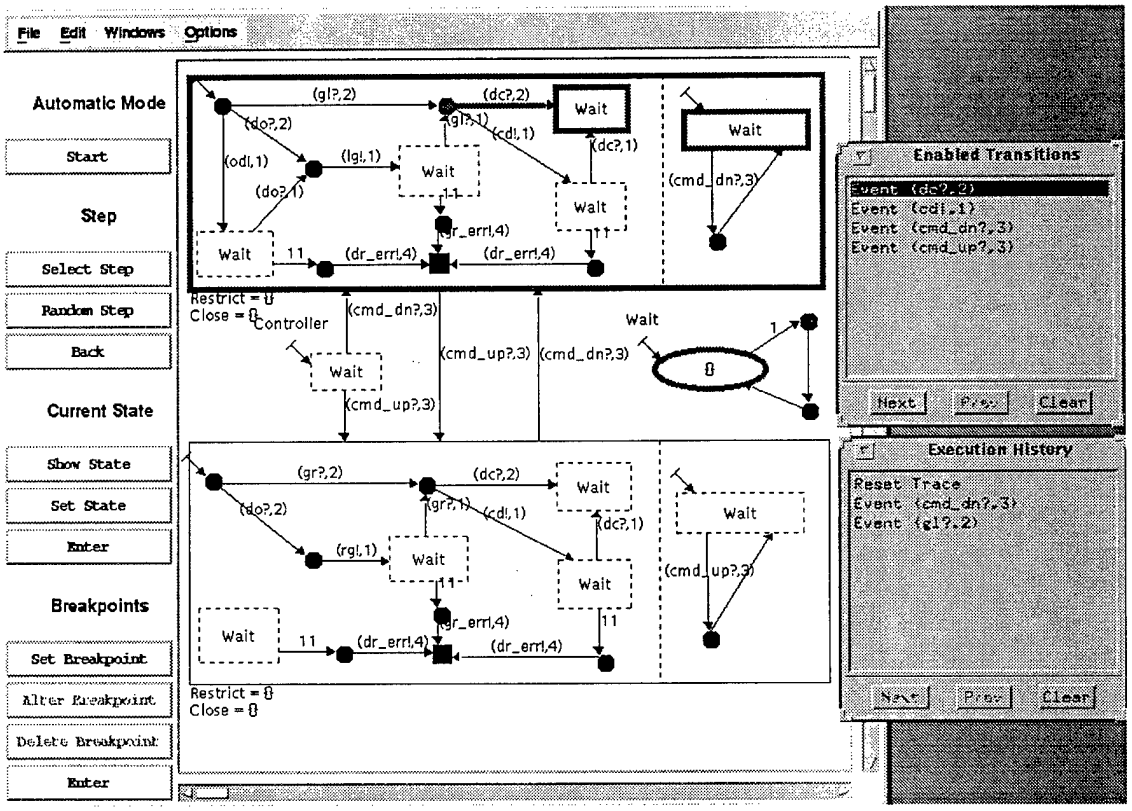

Figure II-2: User Interface of the Simulator

II-3 VERSA Tool

VERSA (**V**erification, **E**xecution and **R**ewrite **S**ystem for **A**CSR) tool was developed by the Real-Time Systems Group at University of Pennsylvania as a vehicle for research in formal methods. We have chosen VERSA as a back-end verification engine for PARAGON, since it supports a compatible formalism. The language of VERSA is a real-time process algebra ACSR. ACSR terms have a one-to-one correspondence with expressions in GCSR, the input language of PARAGON. Automatic translations between the two is possible. For this reason, the natural design decision for us was to take an existing tool and enhance it for our purposes instead of creating a new verification subsystem from scratch.

The three main features of the VERSA system are: syntax checking, compilation, and semantic analysis of ACSR specifications. Compilation uses the semantic rules of ACSR to translate an ACSR specification into its underlying labeled transition system (LTS) representation. Semantic analysis routines of VERSA consist of three major areas: state space exploration, equivalence testing, and interactive execution. State space exploration, equivalence testing and interactive execution operate on the LTS representation of the system being analyzed.

State space exploration analysis can be used to determine key properties of a system's LTS. These include

1. number of states and transitions;
2. presence of deadlocked states;
3. states capable of *Zeno* behaviors (i.e. infinite sequences of instantaneous events, preventing the time from progressing);
4. states that require synchronization to take place before time can progress;
5. reachability of specific externally observable events.

Process equivalence can be tested using a number of different notions of equivalence that behaviorally relate different implementations of the same system, possibly with different levels of abstraction. The interactive execution feature of VERSA allows user-directed execution of ACSR specifications on the LTS level through a rudimentary simulator.

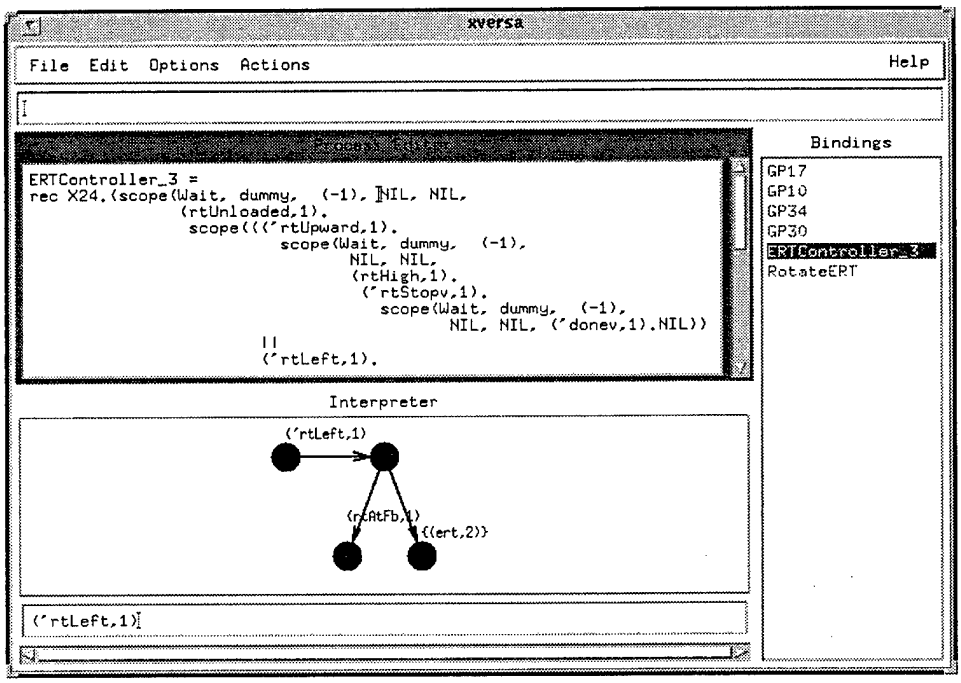

Figure II-3: Graphical User Interface to VERSA

Functionality of VERSA is made available to the user via a graphical user interface shown in Figure II-3. A command-line interface is also available, allowing the user to perform verification from a terminal without windowing capabilities.

In the course of the current project, we modified VERSA code to make it more robust and portable. We performed ports of VERSA to all platforms currently supported by PARAGON (SunOS, Solaris, Linux). We also introduced several heuristics into VERSA aimed at improved performance of VERSA verification algorithms.

II-4 Documentation

Adequate documentation is critical to commercial success of any software tool. We have developed a number of documents describing different aspects of the developed toolset. Some of the documents accompany this report as deliverable items under the present contract.

PARAGON User Manual. The manual [13] is a detailed description of the toolset functionality. The manual mainly concentrates on tools related to GCSR specifications, while description of VERSA functionality and verification capabilities are delegated to a separate VERSA user manual [5].

PARAGON Programmer's Guide. PARAGON is an open system, which is distributed with its source code. We are planning further extensions to PARAGON and intend to incorporate the latest advances in verification technology. Therefore, it is very important that the internal structure of PARAGON is throughly documented. The programmer's guide [12] presents a detailed description of PARAGON's design through the class hierarchy of its objects. The manual also outlines the coding standards and major design decisions made while working on PARAGON.

Tutorial of PARAGON. It is important that users of the tool know how to use technology that it represents effectively. Many potential users of the tool are practitioners in the field of real-time systems design. Most of them have little, if any, knowledge of formal methods and analysis technologies based on them. While PARAGON strives to hide theoretical complexities of formal verification from end users, it is not always completely possible. In order to guide new users through

major specification and verification techniques used in PARAGON, we provided a tutorial for the toolset [11].

A small but meaningful example has been chosen. Starting from the informal problem statement, we showed how to construct specification and perform analysis. Along with a correct specification we demonstrated an erroneous one, and showed how the error was uncovered during analysis.

VERSA User Manual. The manual accompanies PARAGON User Manual and provides detailed description of functionality of VERSA verification tool. The manual has been developed by the Real-Time System Group of the University of Pennsylvania, where VERSA originated. The manual describes in full detail the syntax of ACSR specifications. ACSR specifications produced by automatic translation of GCSR specifications fully comply to this syntax.

In addition, the manual presents the textual command interface to VERSA. This interface has been largely subsumed by XVERSA interface that provides graphical user interface wrappers for VERSA commands. XVERSA interface is described in the PARAGON User Guide [13].

Chapter III

Research Results

III-1 Main Research Directions

In addition to development of software tools, the work on this project included significant research content. The main directions of research performed in this effort were:

1. establish a direct basis for correctness of PARAGON design specification;
2. extend GCSR language with support for parameterization;
3. design a higher level language for requirement specification.
4. develop a methodology for automatic test generation of ACSR specifications.
5. improve efficiency of verification in the PARAGON tools.

The first direction resulted in development of structural operational semantics for GCSR. Previously, semantics for GCSR constructs was given in terms of its translation into ACSR. That approach was formal and unambiguous, and was quite sufficient as long as analysis of GCSR specifications was performed on their ACSR counterparts. In the development of the GCSR simulator, a more direct approach was taken. Simulation of GCSR specifications is performed directly on GCSR graphical primitives, without any intermediate translations. Therefore, development of a new semantic definition was warranted. We chose to derive the new semantics in the operational form, which for every state of a GCSR specification gives us a set of

transitions possible from that state. Since simulation of a specification amounts to repeatedly selecting a transition among the enabled ones and following it to a new state, operational semantics provide an immediate formal basis for simulation. Results of this work have been presented at IEEE National Aerospace and Electronics Conference (NAECON'97) in Dayton, OH in July 1997 [4].

The need for the second research effort was established through a practical application. Support for parameterization through indexed specifications has been a feature of VERSA system since its inception. However, since such parameterization has a mapping into the non-parameterized language, it was not deemed to be a critical feature. Therefore, in the initial design of GCSR language, performed as a part of Phase I contract, there was no support for parameterization. We found out that this adversely affects the readability of top-level GCSR specifications and increases their size. We could no longer encapsulate individual components of the system in a hierarchy of nodes in a natural way. To remedy this, we extended the definition of GCSR language to provide parameterized constructs. The version has been implemented in the GCSR editor, which now fully supports the enhanced syntax and provides additional consistency checks.

It is often necessary to match a detailed system specification of a system against a more abstract requirement specification. Often it is possible to represent the requirement as another ACSR or GCSR specification, and use one of verification techniques provided by VERSA. This approach allows one to capture requirements that represent safety and some liveness properties. When we want to validate more general requirements, we have to resort to a technique called model checking. In that approach, requirements are specified in a separate language based on a temporal logic. In general logical specifications tend to be difficult to understand. To support model checking without violating the commitment of PARAGON to clarity of specification, we introduced a visual language for property specification internally based on a powerful temporal logic. The underlying complexity is hidden from the end user, however, by an intermediate specification layer. At this level, a formal methods expert provides fragments of specifications that are presented to the end user as intuitive building blocks. This allows to construct frameworks specific to individual application domains easily. We have designed and implemented a prototype environment to construct specifications in this language. The environment interfaces with the internal representation for ACSR specifications in VERSA and can be used as a front-end for model checking algorithms, for which working prototypes are also available.

Results of this research effort were presented at the Second Workshop on High-Assurance Systems Engineering (HASE'97) in Washington, DC in August 1997 [19].

Research on automatic test generation was motivated by the fact that many real-life applications are too large for exhaustive verification to be performed on them. Nevertheless, we are able to perform partial analysis of such systems by running series of tests on them. The theory of test generation for real-time systems with the desired degree of coverage has been put forward in [6]. Case studies using this approach showed its utility in analyzing large distributed real-time systems. An additional advantage of this method is that the set of tests generated for an ACSR specification may later be used for testing an implementation of the same system in hardware or software. Results of this research and case studies were published in [7, 8, 9].

Research aimed at improvement of VERSA verification techniques originated from another practical application of PARAGON toolset. The application, described in the next section, contains a real-time kernel that schedules certain tasks at predefined moments in time. Therefore, it spends significant periods of time waiting for the prescribed moment to arrive. The state space of the problem contains, therefore, long chains of time-passing transitions. We found that we can compress these chains into a single transition preserving all properties of the specification. Combined with some other optimization techniques, this improvement allowed us to reduce memory requirements by more than a factor of 3.

III-2 Recent Publications Related to the Project

During the work on this project, members of the research and development team participated in several conferences devoted to application of formal methods in software engineering. The following presentations were made:

- "A Graphical Language with Formal Semantics for the Specification and Analysis of Real-Time Systems," IEEE Real-Time Systems Symposium, December 1995 [3].
- "The Specification and Schedulability Analysis of Real-Time Systems using ACSR," IEEE Real-Time Systems Symposium, December 1995 [10].
- "Schedulability and Safety Analysis in GCSR," WORDS '96: IEEE 2nd International Workshop on Object-oriented Real-time Dependable Systems, La-

guna Beach, CA, February 1996 [2].

- "XVERSA: an integrated graphical and textual toolset for the specification and analysis of resource-bound real-time systems," International Conference on Computer-Aided Verification (CAV'96), New Brunswick, NJ, July 1996 [15];
- "Testing-Based Analysis of Real-Time System Models," International Test Conference, October 1996 [7].
- "The Integrated Specification and Analysis of Functional, Temporal, and Resource Requirements," Conference on Requirement Engineering '97 [16].
- "PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems," IEEE Aerospace Conference, Aspen, CO, February 1997 [1].
- "Automatic Generation of Tests for Timing Constraints from Requirements," WORDS '97: IEEE 3rd International Workshop on Object-oriented Real-time Dependable Systems, February 1997 [8].
- "A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems," Oxford University, UK, Workshop on Formal Methods, June 1996.
- "A Process Algebraic Approach to the Schedulability Analysis of Real-Time Systems," University of Warwick, UK, June 1996.
- "Automatic Test Generation for the Analysis of a Real-Time System: Case Study," Third IEEE Real-Time Technology and Applications Symposium (RTAS '97), Montreal, Canada, June 1997 [9].
- "Introduction to Formal Methods for Real-Time Systems," Systems Engineering Research Institute (SERI), Korea, July 1997.
- "Introduction to Real-Time Process Algebra and its Applications," Korea University, Seoul, Korea, October 1996.
- "PARAGON: A Paradigm for the Specification, Verification and Testing of Real-Time Systems," Chung-Ang University, Seoul, Korea, October 1996.
- "A Process Algebraic Approach to the Specification and Schedulability Analysis of Real-Time Systems," University of Michigan, Ann Arbor, October 1996.

- "A Process Algebraic Approach to the Specification and Schedulability Analysis of Real-Time Systems," University of Illinois, Urbana-Champaign, October 1996.
- "The Specification and Schedulability Analysis of Real-Time Systems using ACSR," State University of New York at Stony Brook, AT&T/SUNY-SB Workshop on Specification and Verification, November 1995.

Selected articles appear as attachments to this report.

Several demonstrations of the toolset were performed:

- ONR Technology Gathering, Naval Warfare Research Center, Dahlgren, VA, May 1997;
- Workshop on Formal Methods, Oxford University, UK, June 1996;
- WORDS '97: IEEE 3rd International Workshop on Object-oriented Real-time Dependable Systems, February 1997;
- Third IEEE Real-Time Technology and Applications Symposium (RTAS '97), Montreal, Canada, June 1997;
- Systems Engineering Research Institute (SERI), Korea, July 1997.

Chapter IV

Commercialization

We began commercialization efforts for PARAGON even before the work on the toolset was completed. These efforts took the form of case studies using real-life applications obtained from potential clients. Early start is important for two reasons. First, this helps us to establish the customer base by the time PARAGON is ready for marketing. Second, being able to work with large specifications allows us to identify deficiencies of the tool, which may not be evident from standard benchmark applications, earlier in the tool development cycle. We were involved in two projects summarized below.

Display LAN communication protocol. The protocol originated from Northrop-Grumman, who provided us with an informal description of the protocol in the form of textual description and event diagrams. The project was cancelled soon after it had started for reasons beyond our control. However, we were able to construct a prototype specification, which highlighted ambiguities in the provided description. The significance of this project for us was that the work on the prototype specification led us to introduce parameterization into GCSR language. Design of parameterized constructs was guided by our attempts to make the prototype protocol specification intuitive enough.

Redundancy Management System. This effort, which is a joint project with Allied Signal Corp., targets specification and analysis of redundancy management system (RMS) for X-33, the next-generation single-stage-to-orbit spacecraft. A hybrid hardware/software implementation of RMS is undergoing the last stages

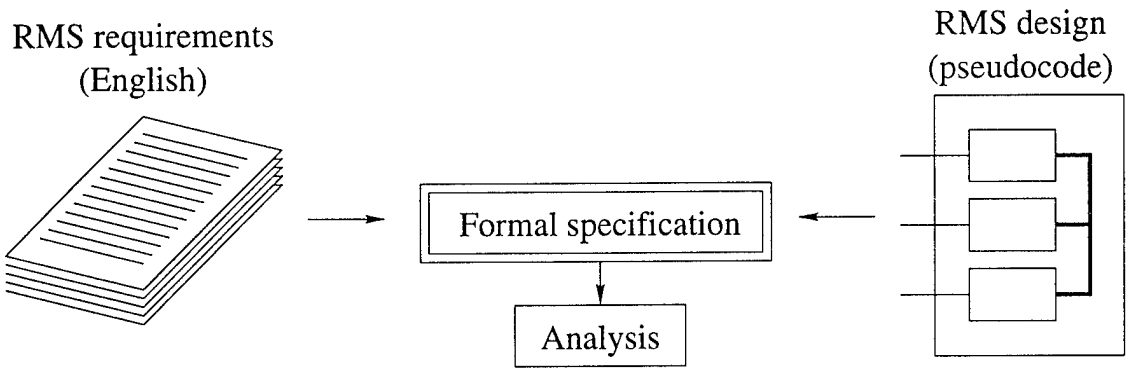

Figure IV-1: Formal Specification of RMS

of design by Allied Signal. They provided us with the actual specifications for RMS and pseudo-code for RMS components used in the design. The diagram in Figure IV-1 summarizes the work performed on this project. The pseudo-code was manually translated into a formal representation, and requirements were formalized as tester processes as described in Section I-4. As with the previous projects, many ambiguities in the requirements were identified in the process of formalization.

Below we present the overview of RMS system to the extent permitted by our non-disclosure agreement with Allied Signal. The system consists of a Fault Tolerant Executive (FTE) and a Cross Channel Data Link (CCDL). The FTE consists of three identical nodes that communicate with each other by means of CCDL. Each FTE node consists of a real-time kernel that controls other components of the node: fault tolerator, voter, synchronizer, and task communicator. Together, these components ensure consistency of data among all non-faulty FTE nodes, as long as no more than one fault occurs at a time. To ensure consistency of data and identify faulty nodes, FTE nodes vote on values of data and status variables in each node.The task communicator performs scheduling of application tasks and exchanges data with applications. The RMS also informs the application if an unresolvable conflict occurs between data values in different nodes.

While work on specification and analysis of FTE nodes continues, preliminary results have already been obtained. We were able to identify several errors in the kernel and fault tolerator pseudo-code that would lead to incorrect behavior of RMS. Fortunately, these errors were present only in the pseudo-code, but were corrected by designers in the actual implementation. However, this indicated an important problem of inconsistency between the specification (pseudo-code) and

implementation of RMS, which we indicated to the designers.

Preliminary results of our early commercialization efforts demonstrate that PARAGON is a toolset capable of handling medium-size commercial applications. Results of this work will give us a solid starting point for commercialization after the devlopment phase is completed.

Our immediate commercialization plans for PARAGON after the completion of the current contract include

- continuation of the RMS project until conclusive results are obtained. These results will be published and used in promotion of PARAGON with other potential customers;
- design of an interface between PARAGON and SCR* toolset [17]. SCR* is a toolset for specification and analysis of requrements, developed at the Naval Research Laboratory under the supervision of Ms. Heitmeyer. SCR* specifications give concise and natural formal representation to high-level requirements. On the other hand, PARAGON provides an efficient way to construct detailed specifications of systems. Of special interest are real-time capabilities of PARAGON. Since SCR* tools are intended for requiement specification, and PARAGON tools are primarily used for design specification, we conclude that the two tools will complement each other and a combined toolset will be a viable commercial-strength CASE environment.

Bibliography

[1] H. Ben-Abdallah, D. Clarke, I. Lee, and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, pages 469–488, Feb 1-8 1997.

[2] H. Ben-Abdallah, Y.-S. Kim, and I. Lee. Schedulability and safety analysis in GCSR. In *Proceedings of WORDS '96: IEEE 2nd International Workshop on Object-oriented Real-time Dependable Systems*, Feb. 1996.

[3] H. Ben-Abdallah, I. Lee, and J.-Y. Choi. A graphical language with formal semantics for the specification and analysis of real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium.* IEEE Computer Society Press, December 1995.

[4] H. Ben-Abdallah, I. Lee, and O. Sokolsky. Operational semantics for visual simulation in PARAGON. In *Proceedings of IEEE National Aerospace and Electronics Conference*, July 1997.

[5] D. Clarke. VERSA: Verification, execution and rewrite system for ACSR. Technical Report MS-CIS-95-34, Department of Computer and Information Science, University of Pennsylvania, Oct. 1995.

[6] D. Clarke. *Testing Real-Time Constraints.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, November 1996.

[7] D. Clarke and I. Lee. Testing-based analysis of real-time system models. In *Proceedings of International Test Conference*, Oct. 1996.

[8] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *Proceedings of WORDS '97: IEEE 3rd International Workshop on Object-oriented Real-time Dependable Systems*, Feb. 1997.

[9] D. Clarke and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *Proceedings of 3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, pages 112–124. IEEE Computer Society Press, June 1997.

[10] D. Clarke, I. Lee, and H.-L. Xie. VERSA: A tool for the specification and analysis of resource-bound real-time systems. *Journal of Computer and Software Engineering*, 3(2), April 1995.

[11] Computer Command and Control Company. *PARAGON Toolset: A tutorial*, Nov. 1996.

[12] Computer Command and Control Company. *PARAGON Programmer's Guide*, Sept. 1997.

[13] Computer Command and Control Company. *PARAGON User Manual*, Sept. 1997.

[14] Computer Command and Control Company. *Software Tools for Formal Specification and Verification of Distributed Real-Time Systems: A Final Report*, July 1997. Submitted under ONR SBIR Phase I Contract No. N00014-94-C-0081.

[15] D.Clarke, H. Ben-Abdallah, I. Lee, H.-L. Xie, and O. Sokolsky. XVERSA: an integrated graphical and textual toolset for the specification and analysis of resource-bound real-time systems. In *Proceedings of Computer-Aided Verification '96*, number 1102 in LNCS, pages 402–405. Springer-Verlag, July 1996.

[16] I. L. H. Ben-Abdallah and Y.-S. Kim. The integrated specification and analysis of functional, temporal, and resource requirements. In *Proceedings of International Conference on Requirement Engineering '97*, 1997.

[17] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: Toolset for specifying and analyzing requirements. In *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95)*, pages 109–122, June 1995.

[18] I. Lee, P. Bremond-Gregoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, 82(1), January 1994.

[19] I. Lee and O. Sokolsky. A graphical property specification language. In *Proceedings of 2nd IEEE Workshop on High-Assurance Systems Engineering.* IEEE Computer Society Press, August 1997.